# Install

<https://docs.docker.com/install/linux/docker-ce/ubuntu/#install-using-the-repository>

Docker indítás hiba fix:

<https://www.digitalocean.com/community/questions/how-to-fix-docker-got-permission-denied-while-trying-to-connect-to-the-docker-daemon-socket>

GPU hiba fix:

docker: Error response from daemon: linux runtime spec devices: could not select device driver "" with capabilities: [[gpu]].

<http://collabnix.com/introducing-new-docker-cli-api-support-for-nvidia-gpus-under-docker-engine-19-03-0-beta-release/>

# Docker használat

Image build:

[sudo] docker build -t collection/name path-to-Dockerfile

Container indítás

[sudo] docker run imagename

Hasznos kapcsolók:

* -h hostname : ne egy random id-t lássunk host névként belépéskor
* -p port-in-docker:port-on-host : port átírányítás (pl. a host adott portját irányítjuk a docker ssh portjára: -p 2222:22)
* --gpus all : minden gpu engedélyezése
* -v [docker-volume | path-on-host]:[path-in-docker] : docker volume vagy hoszt fájl/könyvtár mountolása
* --mount : kb. mint a -v, csak kulcs-érték párokat fogad. **Ha könyvtárat mountolunk, a célkönyvtár tartalma felülíródik a forráskönyvtáréval.** Alapból read-write, a readonly  kapcsolóval lehet írásvédetten mountolni. Pl:
	+ --mount src=etc,target=/etc,**type=volume** : „etc” nevű docker volume mountolása a /etc könyvtárba. **Ha nincs még ilyen volume, automatikusan létrejön.**
	+ --mount src=/srv/data/gpuapps/home,target=/home,**type=bind** : host könyvtár mountolása egy docker könyvtárba. Az src abszolút path kell, hogy legyen (aktuális könyvtár, pl.: src=$(pwd)/akarmi).

Futó konténerek listázása: docker ps

Leállítás: docker stop [ID-from-ps | name-from-ps] # a stop után **nyomkodhatjuk a tabot**, megjelennek a futó példányok

Docker volume-ok létrehozása: docker volume create volume-name

Docker volume-ok törlése: docker volume rm volume-name . Nekem néha nem működött akkor sem, ha nem futott semmilyen docker konténer. Ez az egy parancs segített: docker system prune

# Perzisztencia

Userek:

Lehetséges megoldások:

1. Minden usert felsorolunk a Dockerfile-ban, a jelszavukkal együtt.
	* Hátrány: utólag hozzáadott userek, megváltozott jelszavak új konténer példány indításakor elvesznek.
2. Shadow és/vagy passwd **fájlok** (/etc/shadow és /etc/passwd) mountolása docker indításakor (ld. feljebb).
	* Előny: már elmentett userek megmaradnak új konténer példány indításakor is. Akár a host gép userei/jelszavai is mappelhetők.
	* Hátrány: a dockerből nem írhatók a mappelt fájlok a linux user és password módosító parancsok működése miatt: <https://stackoverflow.com/questions/33013444/can-not-add-new-user-in-docker-container-with-mounted-etc-passwd-and-etc-shado>
3. Link a shadow/passwd fájlokra, a linkek mountolása: <https://stackoverflow.com/questions/47809350/persistent-etc-passwd-on-a-docker-container>
	* Nekem nem igazán működött, hátha valaki ügyesebb.
4. **Teljes /etc könyvtár mountolása docker volume-ként**, tehát egy docker volume-ba szinkronizáljuk a /etc tartalmát. Ha a docker konténer példányosításakor nem létező docker volume-ot mountolunk, akkor az automatikusan létrejön és a docker volume-ba másolódik a /etc tartalma – tehát dockerből visszafelé is tudjuk szinkronizálni a Dockerfile-ban felvett usereket.
	* Előny: mindkét irányba működik a user/pass szinkronizáció. Futó dockeren belül tudunk új usert felvenni, jelszót váltani stb., a docker konténer újraindításakor a változtatások megmaradnak. Ugyanaz a docker image használható más kurzushoz is, ha más etc volume-ot (és home könyvtárakat, ld. lejjebb) mountolunk fel futtatáskor.
	* Hátrányok:
		1. Kicsit overkill ☺
		2. Ha meglévő docker volume-ot mountolunk a konténer példány indításakor, a Dockerfile-ban felsorolt userek / jelszavak felülíródnak a docker volume-ban lévőkkel. Erre a megoldás leginkább az, hogy a Dockerfile-ba csak a legszükségesebb userek kerüljenek, amik nagyjából véglegesek.
		3. Ha extra parancsokat is szeretnénk user futó docker konténeren belüli létrehozásakor lefuttatni, akkor azt a docker példányon belül kell megtenni. Ez gyakorlatilag csak annyit jelent, hogy nem a Dockerfile-ba kell felvenni az extrákat, hanem egy scriptben odatenni a dockerbe. Pl.:

# (Dockerfile):

COPY ./scripts /root

RUN chmod 700 /root/\*.sh

# dockeren belül

/root/my-usearadd-script.sh

Home könyvtárak, oktatási anyagok

Az elvártnak megfelelően működik (némi korlátozással, ld. lejjebb) pl. az is, ha a teljes /home-ot mountoljuk. Lehetőségek:

* Docker volume-ként. Ebbe belenézni csak dockerből lehet.
* Egy fokkal rugalmasabb, ha host könyvtárat mountolunk, így nem kell a dockerbe belépni, hogy lássuk a hallgatók munkáit, illetve a mountolt fájlok (pl. oktatási anyagok) frissítése is egyszerűbb, nem kell hozzá belépni a dockerbe, sem újraindítani az egészet. A dolognak egyetlen hátránya van: a cg-s home nem használható a mount forrásaként, mivel a userek teljesen összekavarodnak, és a dockeren belülről kizárólag rootként lesz írásjogunk bármilyen mountolt fájlra/könyvtárra. Ha a host gép egy lokális könyvtárát használjuk, akkor minden remekül működik, a host gép könyvtárába (akár a docker futása közben) másolt fájlok megjelennek a dockerben és viszont. Dockeren belül létrehozott fájloknak dockeren belül az lesz a tulajdonosa aki létrehozta, a host könyvtárban pedig a dockert indító user.